Knowledge Representation - W6 Logic & Propositional Logic

Ruairí D. O'Reilly

This week's lab is based on the lecture on Propositional Logic. The goal is to enhance your understanding of logical agents from both a practical and theoretical perspective. By the end of the lab, you should be comfortable representing logical sentences, understanding entailment and propositional KB.

By the end of this lab, you should be comfortable representing logical sentences, evaluating entailment using a propositional knowledge base (KB), and using Python to verify relationships between logical statements.

A. Prep-work

Review the following:

- logic.ipynb (as far as Wumpus World KB)
- logic.py (underlying code, Expr, tt_entails and PropKB will be NB)

I. Background on Logical Entailment

Logical entailment is a key concept in propositional logic. It refers to a relationship where one statement (the conclusion) necessarily follows from one or more other statements (the premises). If the premises are true, the conclusion must also be true.

In logical notation, entailment is denoted using the turnstile symbol ($\models$):

- The premises are written to the left of $\models$.
- The conclusion is written to the right of $\models$.

For example, the statement $A, B \models C$ means that if A and B are true, then C must also be true. The goal of this lab is to investigate how entailment can be verified using a truth table enumeration approach with the function tt_entails.

II. Overview

In this lab, you will make use of the following classes and functions from AIMA's logic.py:

A. Expr Class

The Expr class represents logical expressions symbolically. It enables us to:

- Construct logical sentences using operators such as AND (&), OR (|), NOT (~), IMPLIES ($\Rightarrow$), and BICONDITIONAL ($\Leftrightarrow$).
- Create atomic propositions as well as complex logical statements.
- Perform logical manipulations in a readable and natural way using operator overloading.

B. tt_entails Function

The tt_entails function is used to determine if a given set of premises logically entails a conclusion using a truth table approach. This involves systematically evaluating all possible truth value assignments for the symbols involved in the premises and conclusion to check if the conclusion holds in every model where the premises are true.

III. Exercises

A. Part 1: Using tt_entails to Check Entailment

In this section, we explore how to use the tt_entails function to verify entailment relationships. The examples below illustrate how logical statements can be represented and checked for entailment.
Example: Checking Logical Entailment - We will construct the following logical scenario:
Premises:

- Premise 1: If it is hot, then the air conditioner is on (symbolically: H => AC).
- Premise 2: It is hot (symbolically: H).

Conclusion:

- The air conditioner is on (symbolically: AC).

```python
from logic import Expr, tt_entails

# Define symbols for the logical expressions
H = Expr('H')    # It is hot
AC = Expr('AC')  # The air conditioner is on

# Define the premises using Expr
premise1 = Expr('==>', H, AC)  # Premise 1: If it is hot,
    then the air conditioner is on
premise2 = H                   # Premise 2: It is hot

# Combine the premises using logical AND
premises = premise1 & premise2

# Define the conclusion
conclusion = AC  # Conclusion: The air conditioner is on

# Check if the premises entail the conclusion
result = tt_entails(premises, conclusion)

# Output the result
print(f"Do the premises entail the conclusion? {result}")
```

Expected Output:

Do the premises entail the conclusion? True

Example 1: $A \land B \models A$
Premises: $A \land B$
Conclusion: A
The expected output is True, as A must be true if $A \land B$ is true.

Example 2: $A \lor B \models B \lor A$
Premises: $A \lor B$
Conclusion: $B \lor A$
The expected output is True, as logical OR is

commutative.

Example 3: $A \Rightarrow B \models \neg A \vee B$
Premises: $A \Rightarrow B$
Conclusion: $\neg A \vee B$
The expected output is True, as $A \Rightarrow B$ is logically equivalent to $\neg A \vee B$.

B. Part 2: Understanding Logical Entailment

The following exercises involve constructing logical statements and using tt_entails to determine whether the premises entail a specific conclusion.

Exercise 1: Basic Rain Scenario
Premises:

- Premise 1: If it rains, then the ground is wet. ($R \Rightarrow W$)
- Premise 2: It is raining. (R)

Conclusion: The ground is wet. (W) The expected output is True, as both premises imply that the ground must be wet.

Exercise 2: Logical Entailment with Multiple Conditions
Premises:

- Premise 1: If you study, you will pass the exam. ($Study \Rightarrow Pass$)
- Premise 2: If you pass your assessments, you will get an MSc in AI. ($Pass \Rightarrow MSc$)
- Premise 3: You study. ($Study$)

Conclusion: You will get an MSc in AI. (MSc) The expected output is True, since studying implies passing, and passing implies obtaining the MSc in AI.

C. PropKB

In this exercise, you will get familiar with the PropKB class from by modeling a 3x3 Wumpus World with a pit located at [2,2]. You will need to add all appropriate clauses for the world, including pits and breezes, and construct the knowledge base accordingly. The task is to ensure the world is represented accurately based on the logical relationships between pits and breezes.

World Description:

- The Wumpus World consists of a 3x3 grid.
- There is a pit at square [2,2].
- A breeze is present in any square adjacent to a pit (horizontally or vertically).
- You are given the percepts that there is no breeze in [1,1] and there is a pit at [2,2].

Task:

- Initialize a Propositional Knowledge Base using the PropKB class.
- Define propositional symbols for each square on the grid for both pits and breezes (e.g., P11 for a pit in [1,1], B11 for a breeze in [1,1]).
- Add the appropriate clauses to the knowledge base:
  - No pit in [1,1].
  - There is a pit in [2,2].
  - Breezes exist in squares adjacent to pits, using biconditionals to ensure the correct logical relationships.
- After building the knowledge base, the provided code will generate an ASCII representation of the world.

Hint - Use biconditionals (<=>) to define that a breeze in a square is true if and only if there is a pit in one of its adjacent squares.

```
# Check the KB to see the clauses it contains
print("Clauses in the knowledge base:")
for clause in wumpus_kb.clauses:
    print(clause)

# Print out an ASCII representation of the world
world = [[' ' for _ in range(3)] for _ in range(3)]

# Update each square with the correct symbol: 'P' for pit,
    'B' for breeze, '.' for empty
world[0][0] = 'P' if wumpus_kb.ask_if_true(P11) else 'B' if
    wumpus_kb.ask_if_true(B11) else '.'
world[0][1] = 'P' if wumpus_kb.ask_if_true(P12) else 'B' if
    wumpus_kb.ask_if_true(B12) else '.'
world[0][2] = 'P' if wumpus_kb.ask_if_true(P13) else 'B' if
    wumpus_kb.ask_if_true(B13) else '.'
world[1][0] = 'P' if wumpus_kb.ask_if_true(P21) else 'B' if
    wumpus_kb.ask_if_true(B21) else '.'
world[1][1] = 'P' if wumpus_kb.ask_if_true(P22) else 'B' if
    wumpus_kb.ask_if_true(B22) else '.'
world[1][2] = 'P' if wumpus_kb.ask_if_true(P23) else 'B' if
    wumpus_kb.ask_if_true(B23) else '.'
world[2][0] = 'P' if wumpus_kb.ask_if_true(P31) else 'B' if
    wumpus_kb.ask_if_true(B31) else '.'
world[2][1] = 'P' if wumpus_kb.ask_if_true(P32) else 'B' if
    wumpus_kb.ask_if_true(B32) else '.'
world[2][2] = 'P' if wumpus_kb.ask_if_true(P33) else 'B' if
    wumpus_kb.ask_if_true(B33) else '.'

print("\nASCII representation of the world:")
for row in world:
    print(' '.join(row))
```

Deliverables: Complete the code to tell all necessary clauses to the knowledge base using wumpus_kb.tell(...). Ensure that the logical rules governing the relationships between pits and breezes are correctly represented.

Example Output: If implemented correctly, the output should look like the following ASCII grid (assuming the logical rules are correct):

The ASCII representation of the 3x3 Wumpus World grid, where: - ‘P‘ represents a pit. - ‘B‘ represents a breeze. - ‘.‘ represents an empty square.

Example Output

Below is an example of the expected output:

```
ASCII representation of the world:
. B B
```

```
B P B
B B B
```

IV. Insights and Summary

In this lab, we examined how logical entailment can be tested using Python's AIMA tools. We covered:

- How to represent logical statements using the Expr class.
- How to determine entailment using tt_entails by enumerating all possible truth values.
- Practical examples involving entailment relationships, helping solidify understanding of propositional logic.

Logical entailment is essential for building intelligent agents that can make informed decisions based on a knowledge base. By understanding the relationship between premises and conclusions, we can create systems that effectively infer new knowledge.

V. Lab Questions for Further Exploration

1) Modify the examples provided to create your own logical sentences. Does tt_entails provide the expected results?
2) Add new premises to the examples and explore how entailment changes. For example, add a premise like $MSc \Rightarrow Job$ and check if the conclusion Job follows.
3) Consider how the order of premises might affect entailment. Does the commutativity of logical operators hold in every case?

References